IBM Research

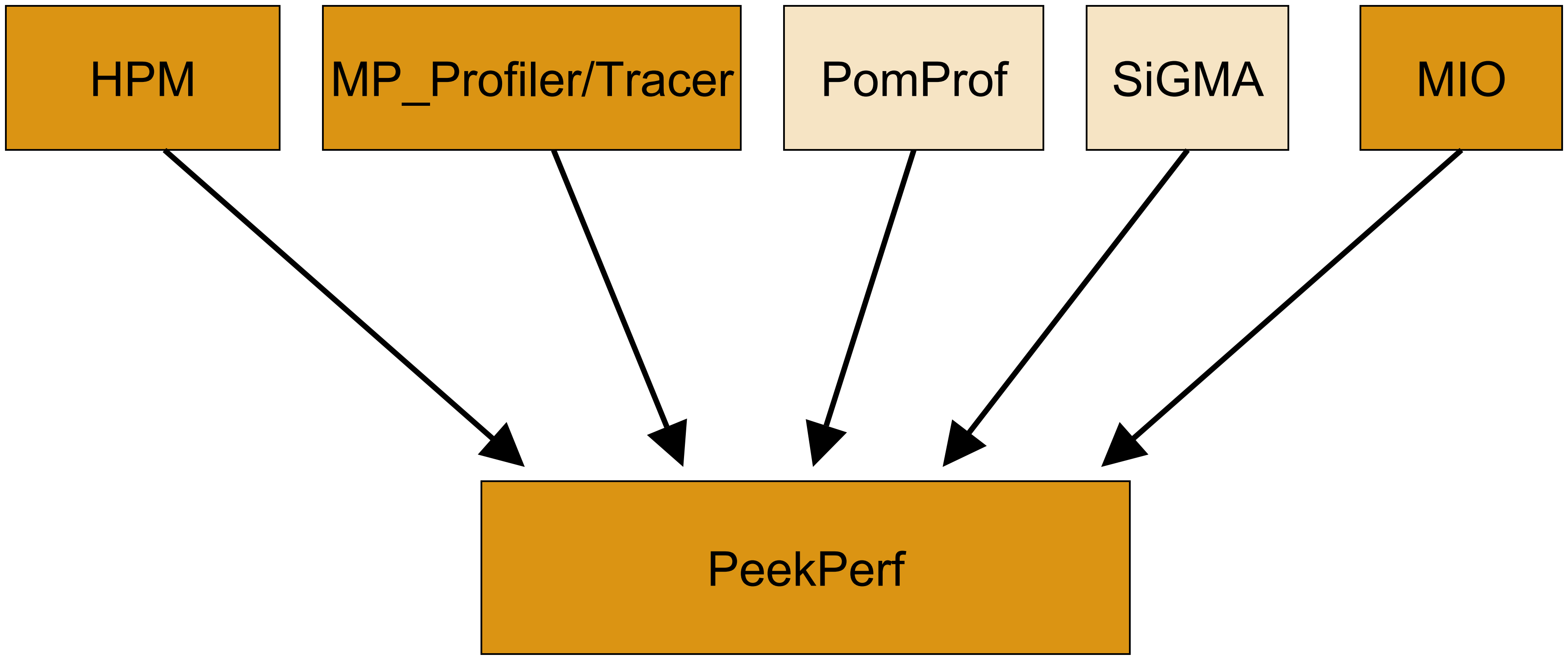
The IBM High Performance Computing Toolkit on BlueGene/L

I-Hsin Chung ihchung@us.ibm.com
Guojing Cong gcong@us.ibm.com
David Klepacki klepacki@us.ibm.com

Outline

- **IBM High Performance Computing Toolkit**
  - MPI performance: MP_Profiler
  - CPU performance: Xprofiler, HPM
  - Modular I/O: MIO
  - Visualization and analysis: PeekPerf
- **Related Tools**
- **Challenges**

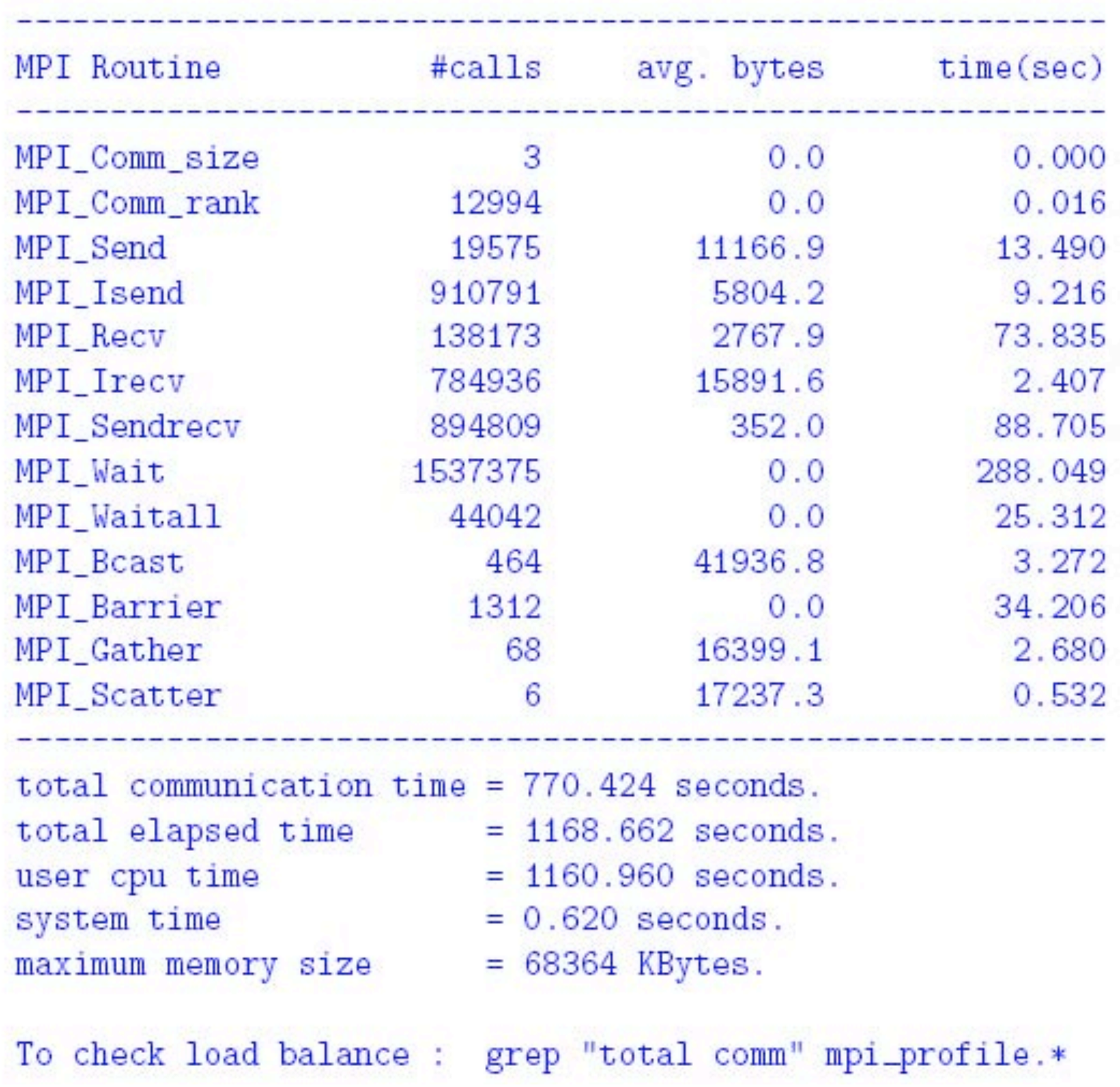
IBM HPCT Overview



Message-Passing Performance:

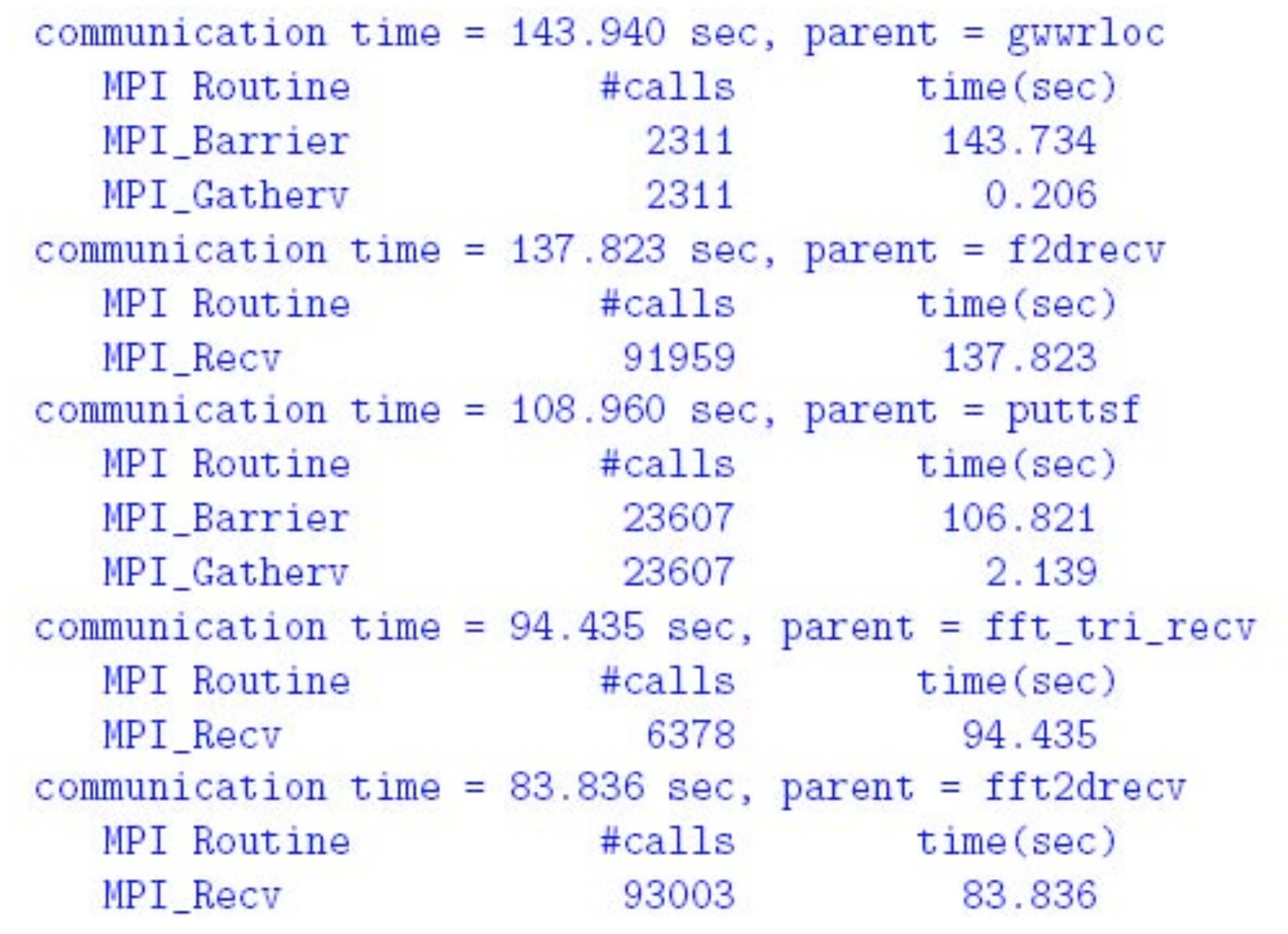
- **MP_Profiler Library**
  - Captures “summary” data for MPI calls
  - Source code traceback
  - User **MUST** call MPI_Finalize() in order to get output files.
  - No changes to source code
    - MUST compile with –g to obtain source line number information
- **MP_Tracer Library**
  - Captures “timestamped” data for MPI calls
  - Source traceback

MP_Profiler Summary Output



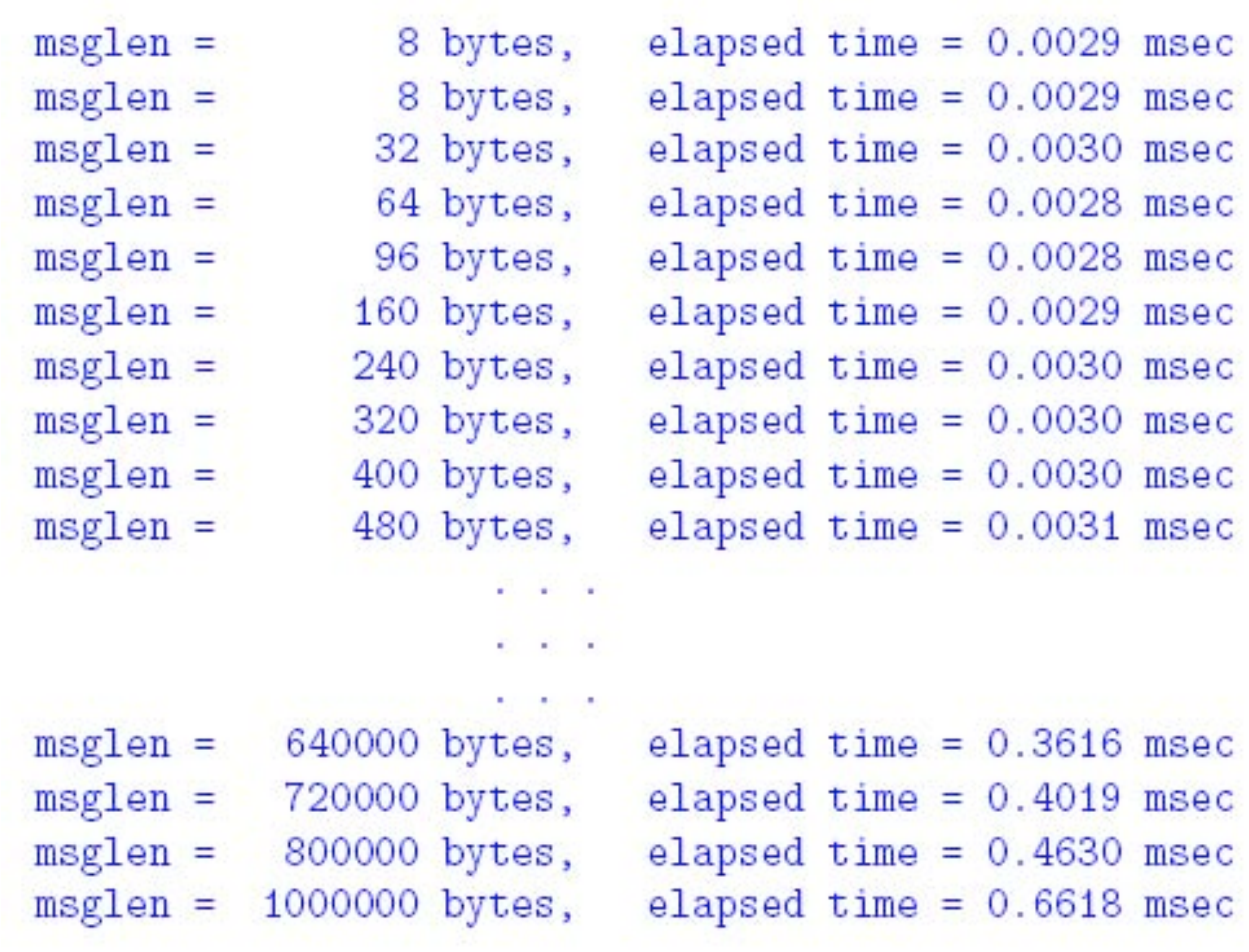
```
-----------------------------------------------------------
MPI Routine              #calls     avg. bytes      time(sec)
-----------------------------------------------------------
MPI_Comm_size                 3            0.0          0.000
MPI_Comm_rank             12994            0.0          0.016
MPI_Send                  19575        11166.9         13.490
MPI_Isend                910791         5804.2          9.216
MPI_Recv                 138173         2767.9         73.835
MPI_Irecv                784936        15891.6          2.407
MPI_Sendrecv             894809          352.0         88.705
MPI_Wait                1537375            0.0        288.049
MPI_Waitall               44042            0.0         25.312
MPI_Bcast                   464        41936.8          3.272
MPI_Barrier                1312            0.0         34.206
MPI_Gather                   68        16399.1          2.680
MPI_Scatter                   6        17237.3          0.532
-----------------------------------------------------------
total communication time = 770.424 seconds.
total elapsed time       = 1168.662 seconds.
user cpu time            = 1160.960 seconds.
system time              = 0.620 seconds.
maximum memory size      = 68364 KBytes.

To check load balance :  grep "total comm" mpi_profile.*
```

MP_Profiler Sample Call Graph Output



```
communication time = 143.940 sec, parent = gwwrloc
   MPI Routine            #calls          time(sec)
   MPI_Barrier              2311           143.734
   MPI_Gatherv              2311             0.206
communication time = 137.823 sec, parent = f2drecv
   MPI Routine            #calls          time(sec)
   MPI_Recv                91959           137.823
communication time = 108.960 sec, parent = puttsf
   MPI Routine            #calls          time(sec)
   MPI_Barrier             23607           106.821
   MPI_Gatherv             23607             2.139
communication time = 94.435 sec, parent = fft_tri_recv
   MPI Routine            #calls          time(sec)
   MPI_Recv                 6378            94.435
communication time = 83.836 sec, parent = fft2drecv
   MPI Routine            #calls          time(sec)
   MPI_Recv                93003            83.836
```

MP_Profiler Message Size Distribution



```
msglen =       8 bytes,   elapsed time = 0.0029 msec
msglen =       8 bytes,   elapsed time = 0.0029 msec
msglen =      32 bytes,   elapsed time = 0.0030 msec
msglen =      64 bytes,   elapsed time = 0.0028 msec
msglen =      96 bytes,   elapsed time = 0.0028 msec
msglen =     160 bytes,   elapsed time = 0.0029 msec
msglen =     240 bytes,   elapsed time = 0.0030 msec
msglen =     320 bytes,   elapsed time = 0.0030 msec
msglen =     400 bytes,   elapsed time = 0.0030 msec
msglen =     480 bytes,   elapsed time = 0.0031 msec
                   . . .
                   . . .
                   . . .
msglen =  640000 bytes,   elapsed time = 0.3616 msec
msglen =  720000 bytes,   elapsed time = 0.4019 msec
msglen =  800000 bytes,   elapsed time = 0.4630 msec
msglen = 1000000 bytes,   elapsed time = 0.6618 msec
```

Hardware Performance Monitor (HPM)

- **Provides comprehensive reports of events that are critical to performance on IBM systems.**
- **Gather critical hardware performance metrics, e.g.**
  - Number of misses on all cache levels
  - Number of floating point instructions executed
  - Number of instruction loads that cause TLB misses
- **Helps to identify and eliminate performance bottlenecks.**

HPM

- **hpmcount**
  - Starts application and provides
    - Wall clock time
    - Hardware performance counter information
    - Resource utilization statistics
- **libhpm**
  - Library for program (including multi-thread) section instrumentation.
- **hpmstat**
  - Provides system wide reports for root

Hpmcount Example

```
bash-2.05a$ hpm_count swim
……  // program output
 hpmcount (V 2.5.4) summary
 Execution time (wall clock time)                         : 7.378159 seconds
 ########  Resource Usage Statistics  ########
 Total amount of time in user mode                        : 0.010000 seconds
Average shared memory use in text segment                 : 672 Kbytes*sec
………
  PM_FPU_FDIV (FPU executed FDIV instruction)             : 0
  PM_FPU_FMA (FPU executed multiply-add instruction)      : 0
  PM_CYC (Processor cycles)                               : 54331072
  PM_FPU_STF (FPU executed store instruction)             : 2172
  PM_INST_CMPL (Instructions completed)                   : 17928229
  Utilization rate                                        : 0.446 %
  Total load and store operations                         : 0.004 M
  MIPS                                                    : 2.140
  Instructions per cycle                                  : 0.330
```

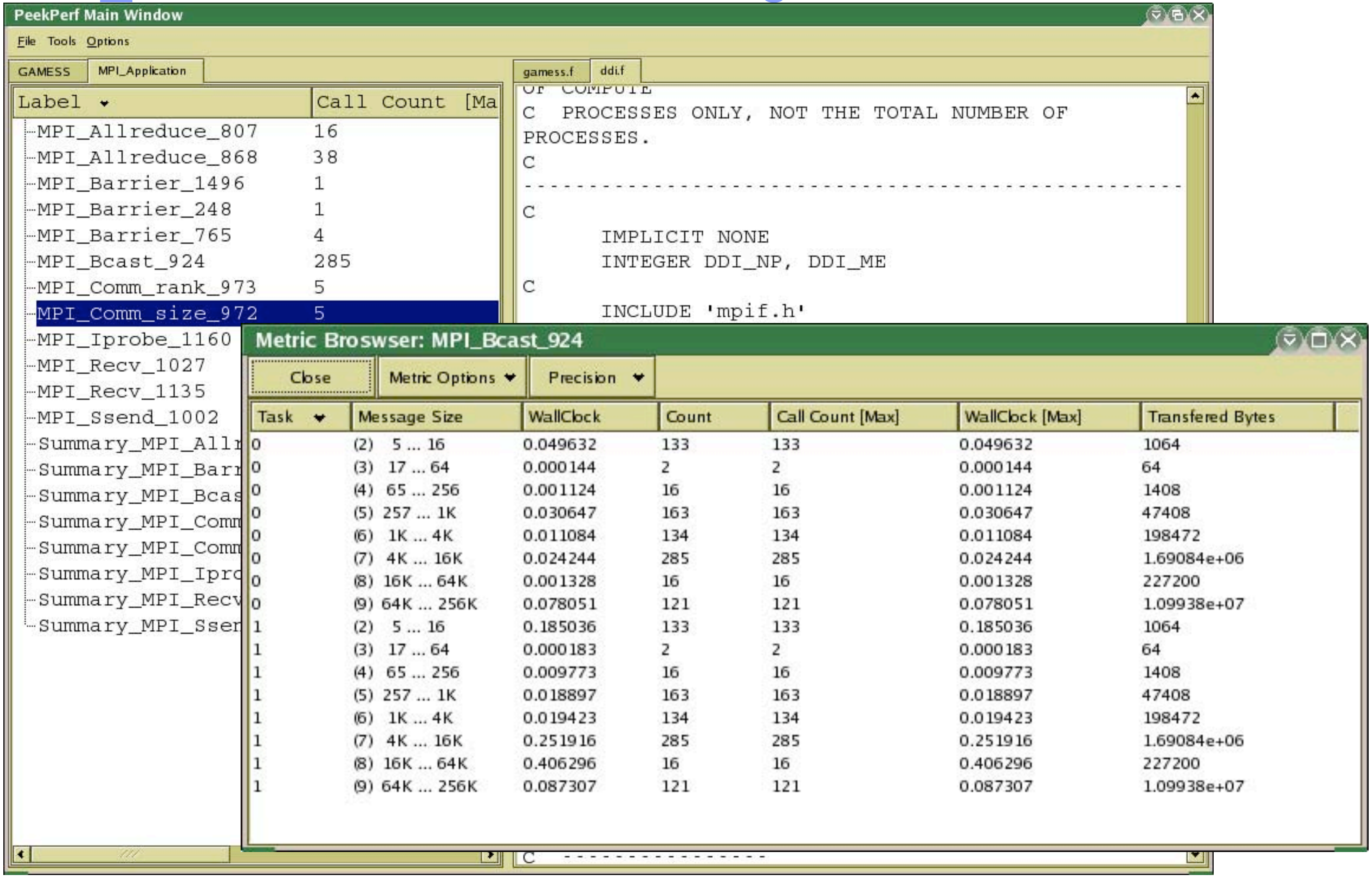
Libhpm Example

```
 call f_hpmstart( 30, "Loop 300" )
C$OMP PARALLELDO
C$OMP&SHARED (ALPHA,M,N,U,V,P,UNEW,VNEW,PNEW,UOLD,VOLD,POLD)
C$OMP&SHARED (JS,JE)
C$OMP&PRIVATE (I,J)
      DO 300 J=js,je
      DO 300 I=1,M
      UOLD(I,J) = U(I,J)+ALPHA*(UNEW(I,J)-2.*U(I,J)+UOLD(I,J))
       ……
       P(I,J) = PNEW(I,J)
  300 CONTINUE
      call f_hpmstop( 30 )
```

Modular I/O (MIO)

- **Addresses the need of application-level optimization for I/O.**
- **Analyze and tune I/O at the application level**
  - For example, when an application exhibits the I/O pattern of sequential reading of large files
  - MIO
    - Detects the behavior
    - Invokes its asynchronous prefetching module to prefetch user data.
- **Planned Integration into HPC Toolkit with PeekPerf capabilities**
  - Source code traceback
  - Future capability for dynamic I/O instrumentation

MP_Profiler Visualization Using PeekPerf

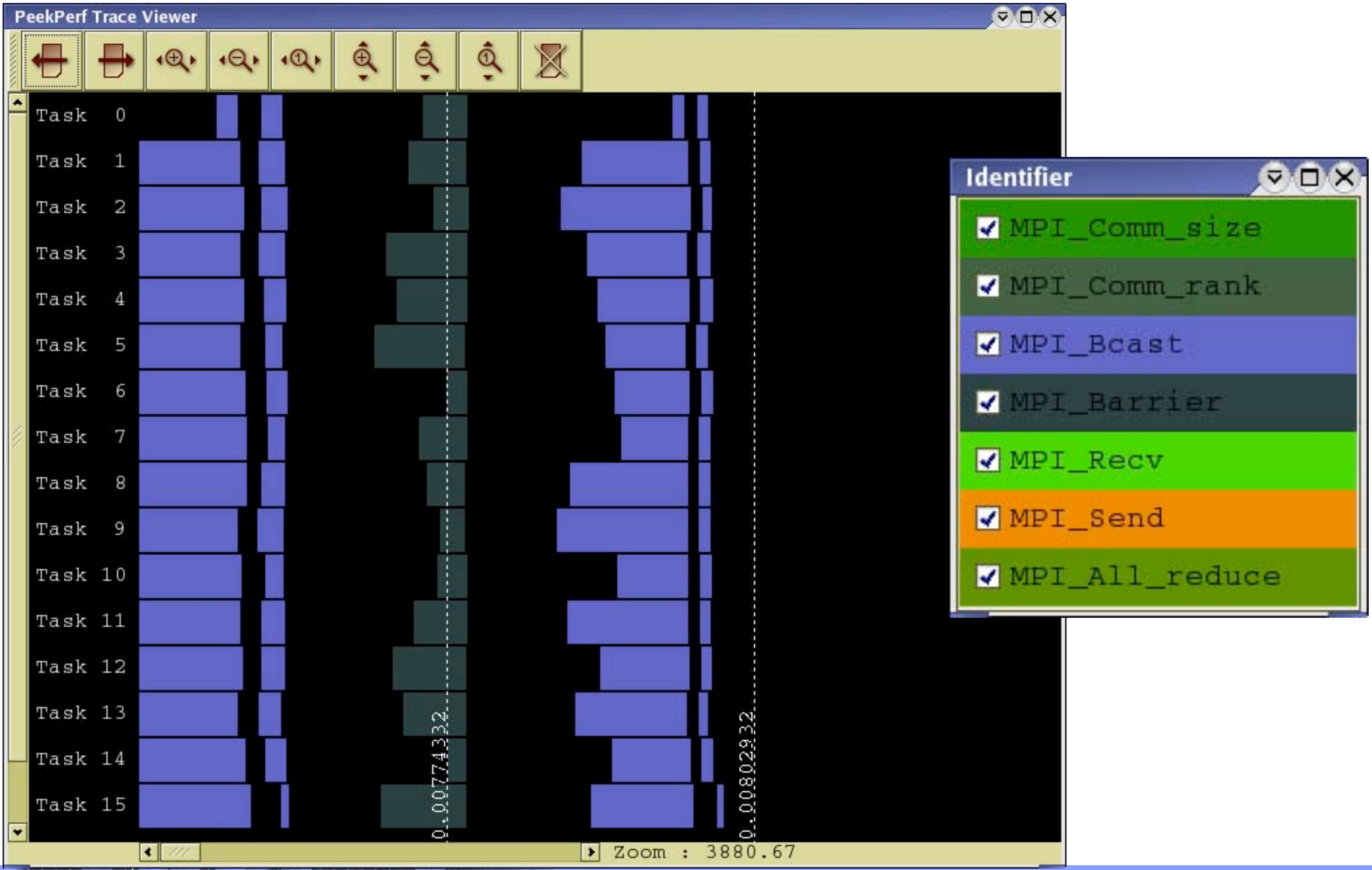


Label	Call Count [Ma
MPI_Allreduce_807	16
MPI_Allreduce_868	38
MPI_Barrier_1496	1
MPI_Barrier_248	1
MPI_Barrier_765	4
MPI_Bcast_924	285
MPI_Comm_rank_973	5
MPI_Comm_size_972	5
MPI_Iprobe_1160	
MPI_Recv_1027	
MPI_Recv_1135	
MPI_Ssend_1002	
Summary_MPI_Allr	
Summary_MPI_Barr	
Summary_MPI_Bcas	
Summary_MPI_Comm	
Summary_MPI_Comm	
Summary_MPI_Ipro	
Summary_MPI_Recv	
Summary_MPI_Ssen	

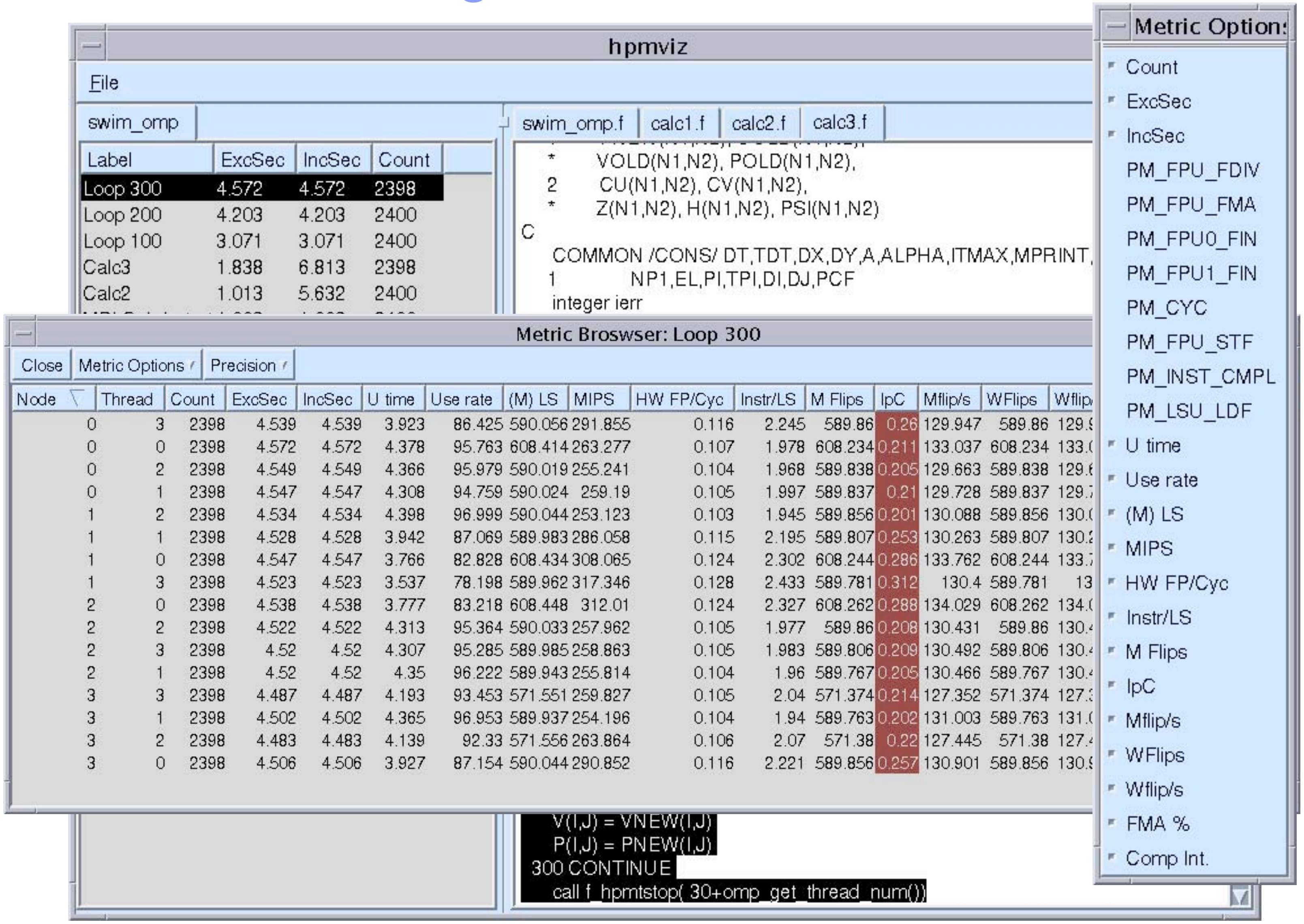
```
C    PROCESSES ONLY, NOT THE TOTAL NUMBER OF
PROCESSES.
C
-------------------------------------------------------------
C
         IMPLICIT NONE
         INTEGER DDI_NP, DDI_ME
C
         INCLUDE 'mpif.h'
C    ----------------
```

Task	Message Size	WallClock	Count	Call Count [Max]	WallClock [Max]	Transfered Bytes
0	(2) 5 ... 16	0.049632	133	133	0.049632	1064
0	(3) 17 ... 64	0.000144	2	2	0.000144	64
0	(4) 65 ... 256	0.001124	16	16	0.001124	1408
0	(5) 257 ... 1K	0.030647	163	163	0.030647	47408
0	(6) 1K ... 4K	0.011084	134	134	0.011084	198472
0	(7) 4K ... 16K	0.024244	285	285	0.024244	1.69084e+06
0	(8) 16K ... 64K	0.001328	16	16	0.001328	227200
0	(9) 64K ... 256K	0.078051	121	121	0.078051	1.09938e+07
1	(2) 5 ... 16	0.185036	133	133	0.185036	1064
1	(3) 17 ... 64	0.000183	2	2	0.000183	64
1	(4) 65 ... 256	0.009773	16	16	0.009773	1408
1	(5) 257 ... 1K	0.018897	163	163	0.018897	47408
1	(6) 1K ... 4K	0.019423	134	134	0.019423	198472
1	(7) 4K ... 16K	0.251916	285	285	0.251916	1.69084e+06
1	(8) 16K ... 64K	0.406296	16	16	0.406296	227200
1	(9) 64K ... 256K	0.087307	121	121	0.087307	1.09938e+07

MP_Tracer Visualization Using PeekPerf



HPM Visualization Using PeekPerf



Label	ExcSec	IncSec	Count
Loop 300	4.572	4.572	2398
Loop 200	4.203	4.203	2400
Loop 100	3.071	3.071	2400
Calc3	1.838	6.813	2398
Calc2	1.013	5.632	2400

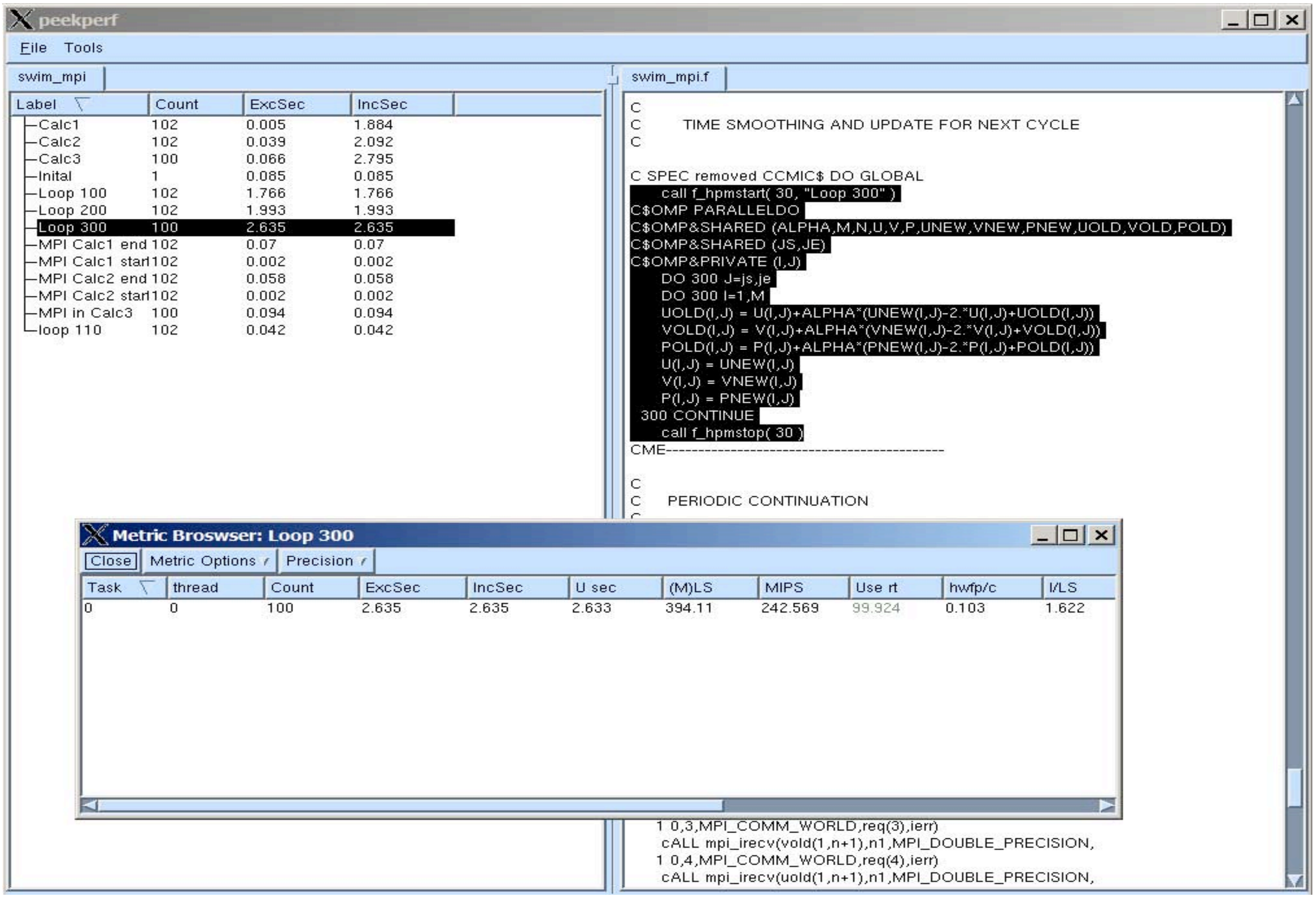
```
     *      VOLD(N1,N2), POLD(N1,N2),
     2      CU(N1,N2), CV(N1,N2),
     *      Z(N1,N2), H(N1,N2), PSI(N1,N2)
C
      COMMON /CONS/ DT,TDT,DX,DY,A,ALPHA,ITMAX,MPRINT,
     1        NP1,EL,PI,TPI,DI,DJ,PCF
      integer ierr
```

```
      V(I,J) = VNEW(I,J)
      P(I,J) = PNEW(I,J)
  300 CONTINUE
      call f_hpmtstop( 30+omp_get_thread_num())
```

Node	Thread	Count	ExcSec	IncSec	U time	Use rate	(M) LS	MIPS	HW FP/Cyc	Instr/LS	M Flips	IpC	Mflip/s	WFlips	Wflip
0	3	2398	4.539	4.539	3.923	86.425	590.056	291.855	0.116	2.245	589.86	0.26	129.947	589.86	129.
0	0	2398	4.572	4.572	4.378	95.763	608.414	263.277	0.107	1.978	608.234	0.211	133.037	608.234	133.
0	2	2398	4.549	4.549	4.366	95.979	590.019	255.241	0.104	1.968	589.838	0.205	129.663	589.838	129.
0	1	2398	4.547	4.547	4.308	94.759	590.024	259.19	0.105	1.997	589.837	0.21	129.728	589.837	129.
1	2	2398	4.534	4.534	4.398	96.999	590.044	253.123	0.103	1.945	589.856	0.201	130.088	589.856	130.
1	1	2398	4.528	4.528	3.942	87.069	589.983	286.058	0.115	2.195	589.807	0.253	130.263	589.807	130.
1	0	2398	4.547	4.547	3.766	82.828	608.434	308.065	0.124	2.302	608.244	0.286	133.762	608.244	133.
1	3	2398	4.523	4.523	3.537	78.198	589.962	317.346	0.128	2.433	589.781	0.312	130.4	589.781	13
2	0	2398	4.538	4.538	3.777	83.218	608.448	312.01	0.124	2.327	608.262	0.288	134.029	608.262	134.
2	2	2398	4.522	4.522	4.313	95.364	590.033	257.962	0.105	1.977	589.86	0.208	130.431	589.86	130.
2	3	2398	4.52	4.52	4.307	95.285	589.985	258.863	0.105	1.983	589.806	0.209	130.492	589.806	130.
2	1	2398	4.52	4.52	4.35	96.222	589.943	255.814	0.104	1.96	589.767	0.205	130.466	589.767	130.
3	3	2398	4.487	4.487	4.193	93.453	571.551	259.827	0.105	2.04	571.374	0.214	127.352	571.374	127.
3	1	2398	4.502	4.502	4.365	96.953	589.937	254.196	0.104	1.94	589.763	0.202	131.003	589.763	131.
3	2	2398	4.483	4.483	4.139	92.33	571.556	263.864	0.106	2.07	571.38	0.22	127.445	571.38	127.
3	0	2398	4.506	4.506	3.927	87.154	590.044	290.852	0.116	2.221	589.856	0.257	130.901	589.856	130.

- Count
- ExcSec
- IncSec
- PM_FPU_FDIV
- PM_FPU_FMA
- PM_FPU0_FIN
- PM_FPU1_FIN
- PM_CYC
- PM_FPU_STF
- PM_INST_CMPL
- PM_LSU_LDF
- U time
- Use rate
- (M) LS
- MIPS
- HW FP/Cyc
- Instr/LS
- M Flips
- IpC
- Mflip/s
- WFlips
- Wflip/s
- FMA %
- Comp Int.

Check Performance Counter



Label	Count	ExcSec	IncSec
Calc1	102	0.005	1.884
Calc2	102	0.039	2.092
Calc3	100	0.066	2.795
Inital	1	0.085	0.085
Loop 100	102	1.766	1.766
Loop 200	102	1.993	1.993
Loop 300	100	2.635	2.635
MPI Calc1 end	102	0.07	0.07
MPI Calc1 start	102	0.002	0.002
MPI Calc2 end	102	0.058	0.058
MPI Calc2 start	102	0.002	0.002
MPI in Calc3	100	0.094	0.094
loop 110	102	0.042	0.042

```
C
C     TIME SMOOTHING AND UPDATE FOR NEXT CYCLE
C

C SPEC removed CCMIC$ DO GLOBAL
      call f_hpmstart( 30, "Loop 300" )
C$OMP PARALLELDO
C$OMP&SHARED (ALPHA,M,N,U,V,P,UNEW,VNEW,PNEW,UOLD,VOLD,POLD)
C$OMP&SHARED (JS,JE)
C$OMP&PRIVATE (I,J)
      DO 300 J=js,je
      DO 300 I=1,M
      UOLD(I,J) = U(I,J)+ALPHA*(UNEW(I,J)-2.*U(I,J)+UOLD(I,J))
      VOLD(I,J) = V(I,J)+ALPHA*(VNEW(I,J)-2.*V(I,J)+VOLD(I,J))
      POLD(I,J) = P(I,J)+ALPHA*(PNEW(I,J)-2.*P(I,J)+POLD(I,J))
      U(I,J) = UNEW(I,J)
      V(I,J) = VNEW(I,J)
      P(I,J) = PNEW(I,J)
  300 CONTINUE
      call f_hpmstop( 30 )
CME-------------------------------------------

C
C     PERIODIC CONTINUATION
```

```
     1 0,3,MPI_COMM_WORLD,req(3),ierr)
      cALL mpi_irecv(vold(1,n+1),n1,MPI_DOUBLE_PRECISION,
     1 0,4,MPI_COMM_WORLD,req(4),ierr)
      cALL mpi_irecv(uold(1,n+1),n1,MPI_DOUBLE_PRECISION,
```

Task	thread	Count	ExcSec	IncSec	U sec	(M)LS	MIPS	Use rt	hwfp/c	I/LS
0	0	100	2.635	2.635	2.633	394.11	242.569	99.924	0.103	1.622

Xprofiler

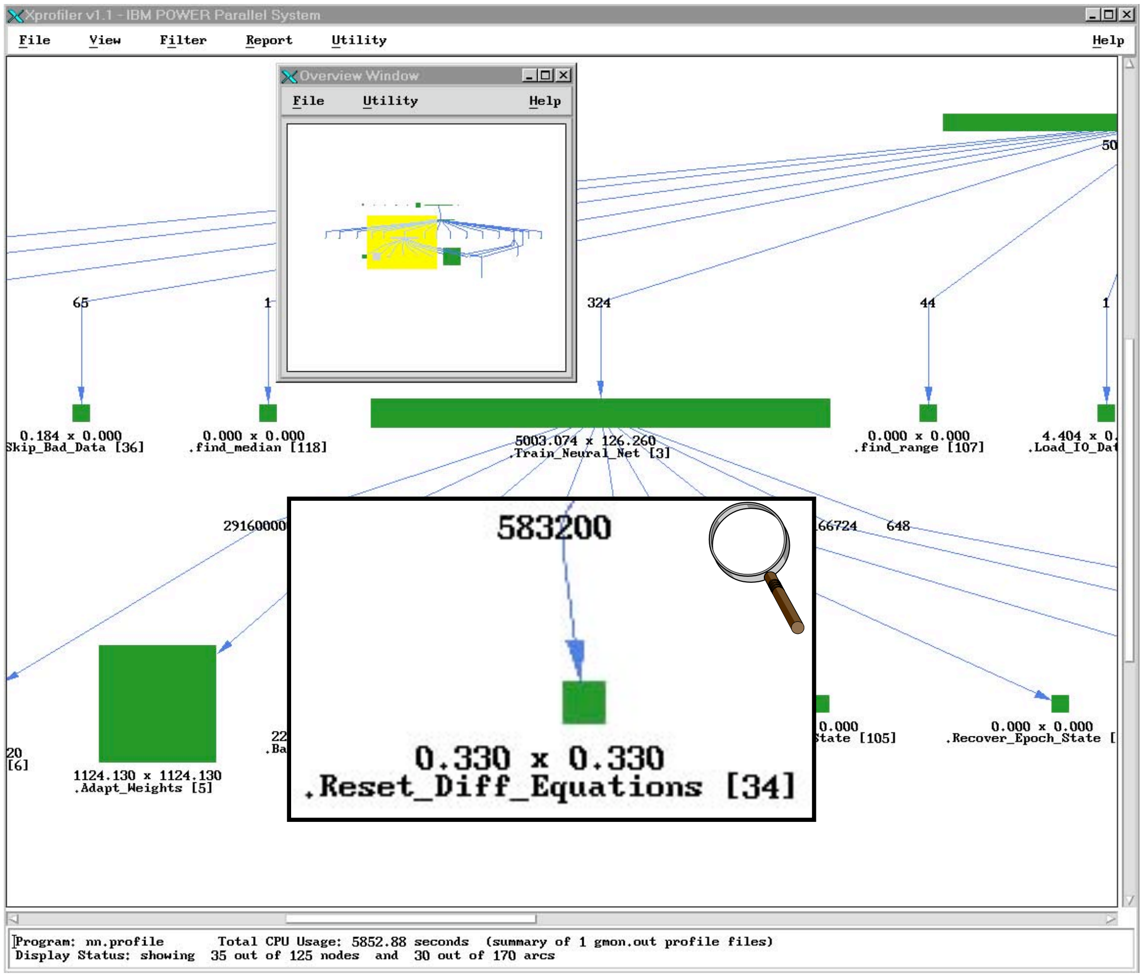
- **CPU profiling tool, extension of gprof**
  - can be used to profile both serial and parallel applications.
- **Graphical display of the call graphs of the application**
- **Provides quick access to the profiled data**
- **Helps users identify the CPU-intensive functions**

Running Xprofiler

- **Compile the program with –pg**
- **Run the program**
- **gmon.out file is generated (MPI applications generate gmon.out.1, …, gmon.out.n)**
- **Run Xprofiler**

Xprofiler: Main Display

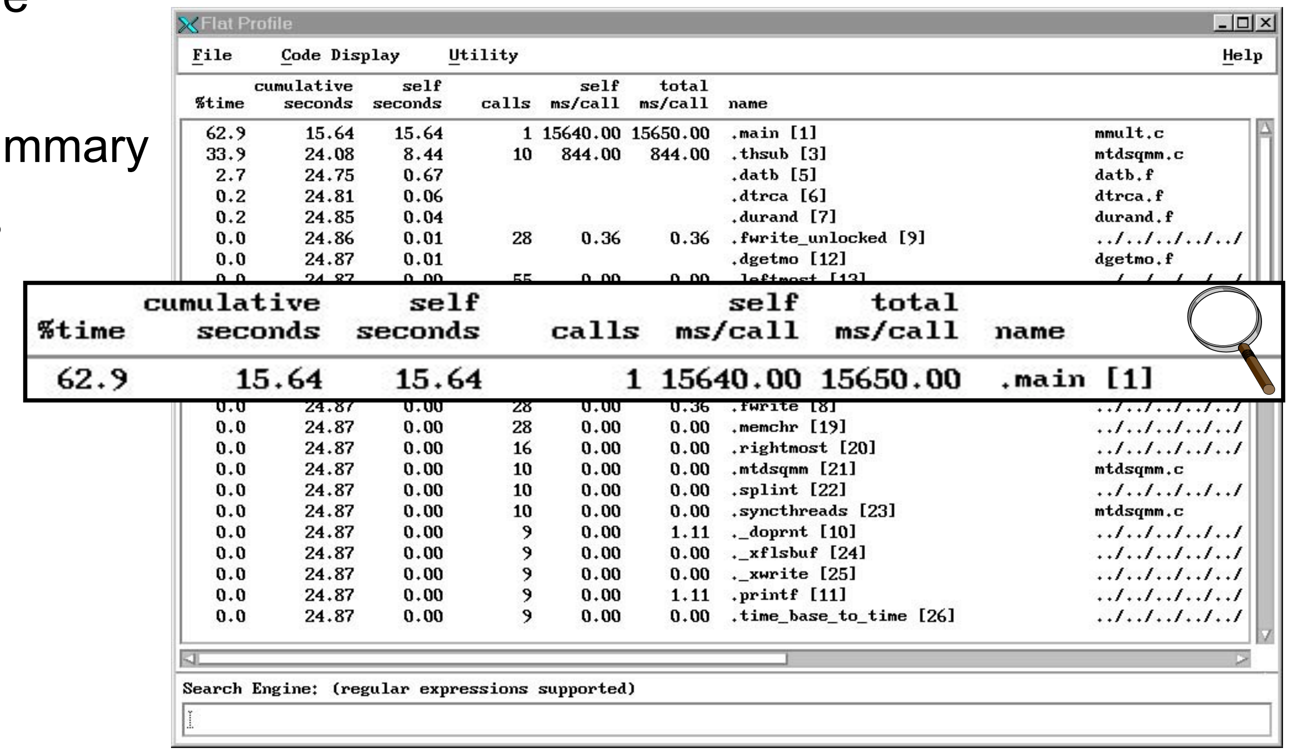
- Width of a bar: time including called routines
- Height of a bar: time excluding called routines
- Call arrows labeled with number of calls
- Overview window for easy navigation (View → Overview)



```
Program: nn.profile        Total CPU Usage: 5852.88 seconds  (summary of 1 gmon.out profile files)
Display Status: showing  35 out of 125 nodes  and  30 out of 170 arcs
```

Xprofiler: Flat Profile

- **Menu** `Report` **provides usual gprof reports plus some extra ones**
  - Flat Profile
  - Call Graph Profile
  - Function Index
  - Function Call Summary
  - Library Statistics



```
        cumulative     self              self      total
%time      seconds   seconds    calls  ms/call   ms/call  name
  62.9       15.64     15.64        1 15640.00 15650.00  .main [1]                mmult.c
  33.9       24.08      8.44       10   844.00   844.00  .thsub [3]               mtdsqmm.c
   2.7       24.75      0.67                             .datb [5]                datb.f
   0.2       24.81      0.06                             .dtrca [6]               dtrca.f
   0.2       24.85      0.04                             .durand [7]              durand.f
   0.0       24.86      0.01       28     0.36     0.36  .fwrite_unlocked [9]     ../../../../../
   0.0       24.87      0.01                             .dgetmo [12]             dgetmo.f
   0.0       24.87      0.00       55     0.00     0.00  .leftmost [13]           ../../../../../
```

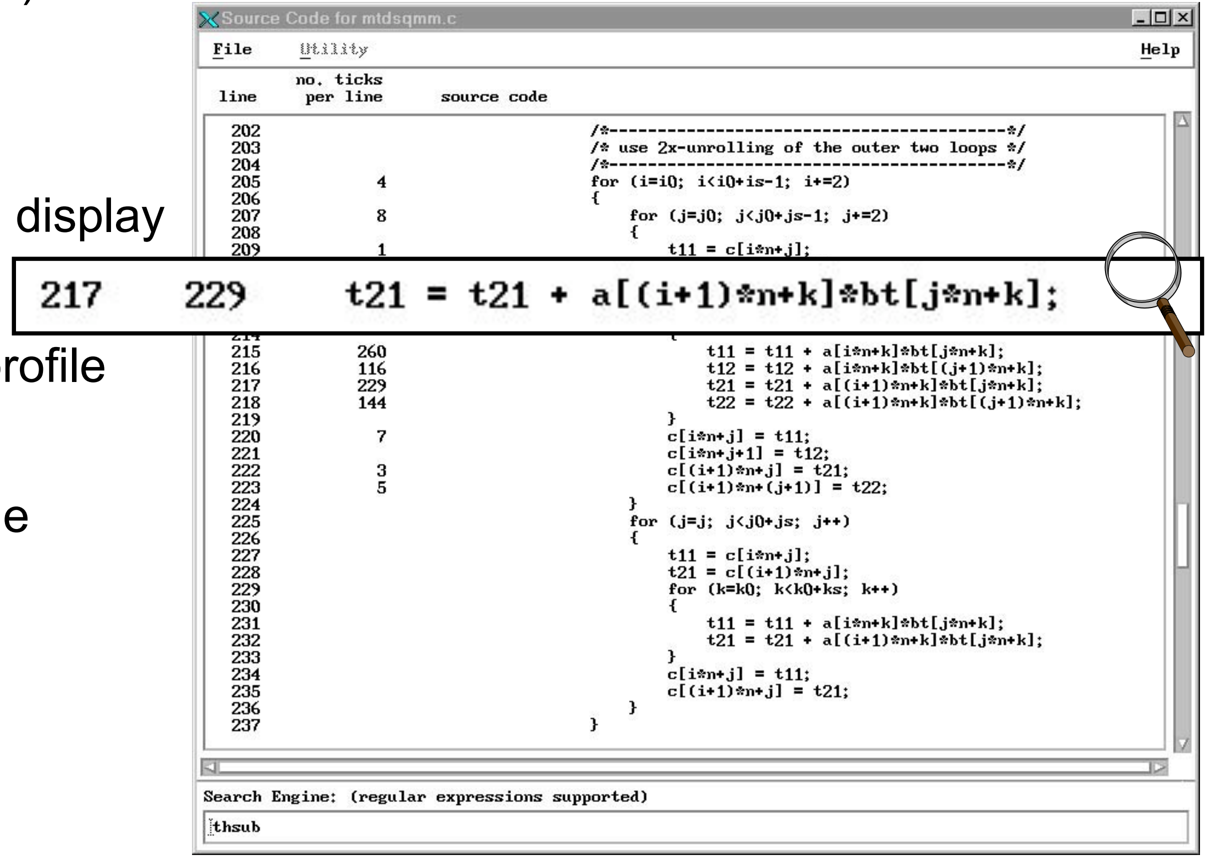
```
        cumulative     self              self      total
%time      seconds   seconds    calls  ms/call   ms/call  name
  62.9       15.64     15.64        1 15640.00 15650.00  .main [1]
```

```
   0.0       24.87      0.00       28     0.00     0.36  .fwrite [8]              ../../../../../
   0.0       24.87      0.00       28     0.00     0.00  .memchr [19]             ../../../../../
   0.0       24.87      0.00       16     0.00     0.00  .rightmost [20]          ../../../../../
   0.0       24.87      0.00       10     0.00     0.00  .mtdsqmm [21]            mtdsqmm.c
   0.0       24.87      0.00       10     0.00     0.00  .splint [22]             ../../../../../
   0.0       24.87      0.00       10     0.00     0.00  .syncthreads [23]        mtdsqmm.c
   0.0       24.87      0.00        9     0.00     1.11  ._doprnt [10]            ../../../../../
   0.0       24.87      0.00        9     0.00     0.00  ._xflsbuf [24]           ../../../../../
   0.0       24.87      0.00        9     0.00     0.00  ._xwrite [25]            ../../../../../
   0.0       24.87      0.00        9     0.00     1.11  .printf [11]             ../../../../../
   0.0       24.87      0.00        9     0.00     0.00  .time_base_to_time [26]  ../../../../../
```

Search Engine: (regular expressions supported)

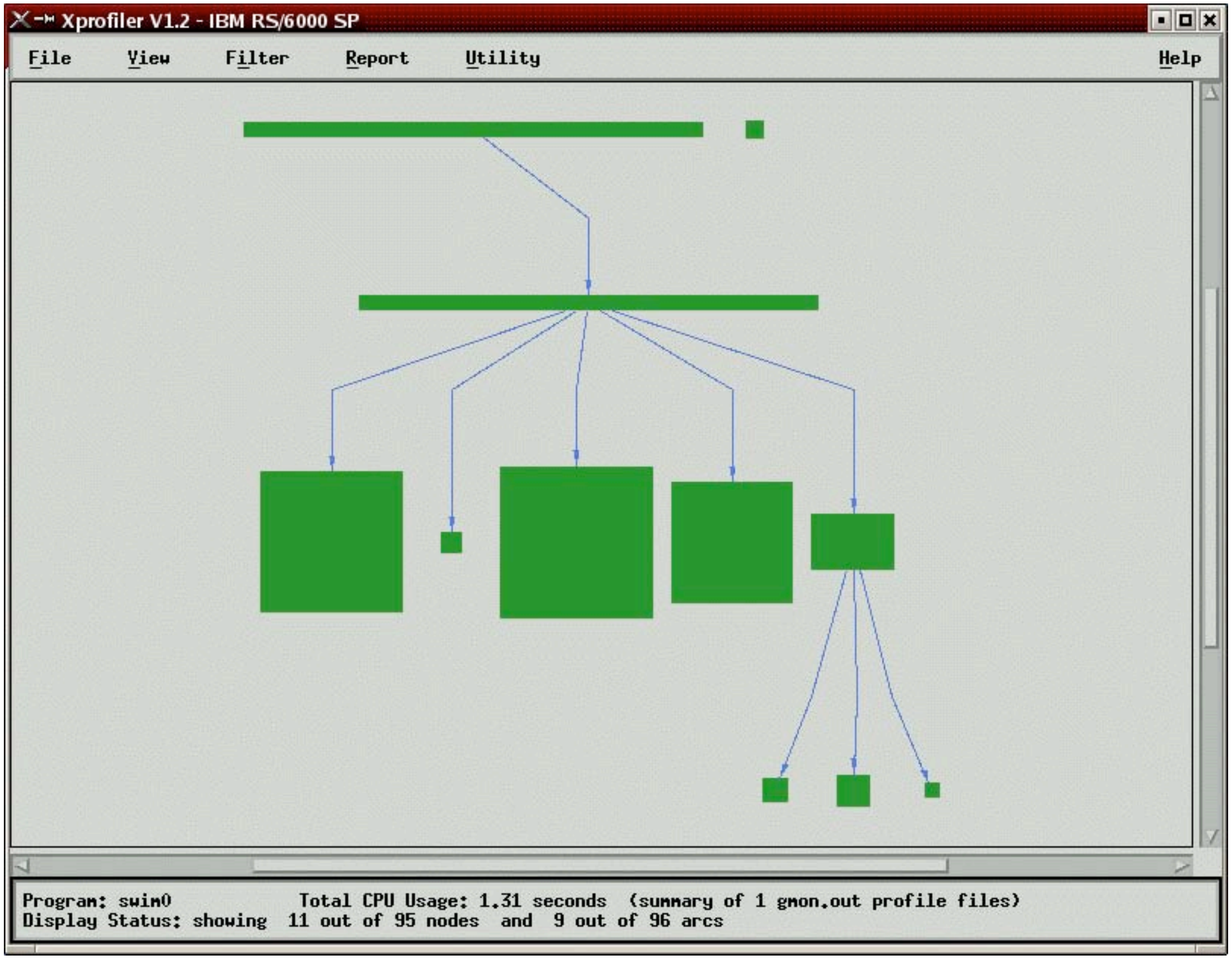
Xprofiler: Source Code Window

- Source code Window displays source code with time profile (resolution 1 tick = 0.01 sec)
- Access
  - Select function in main display
  - → context menu
  - Select function in flat profile
  - → Code Display
  - → Show Source Code



```
        no. ticks
line    per line      source code

 202                              /*-----------------------------------------*/
 203                              /* use 2x-unrolling of the outer two loops */
 204                              /*-----------------------------------------*/
 205          4                   for (i=i0; i<i0+is-1; i+=2)
 206                              {
 207          8                       for (j=j0; j<j0+js-1; j+=2)
 208                                  {
 209          1                           t11 = c[i*n+j];

 217    229       t21 = t21 + a[(i+1)*n+k]*bt[j*n+k];

 215        260                                   t11 = t11 + a[i*n+k]*bt[j*n+k];
 216        116                                   t12 = t12 + a[i*n+k]*bt[(j+1)*n+k];
 217        229                                   t21 = t21 + a[(i+1)*n+k]*bt[j*n+k];
 218        144                                   t22 = t22 + a[(i+1)*n+k]*bt[(j+1)*n+k];
 219                                      }
 220          7                           c[i*n+j] = t11;
 221                                      c[i*n+j+1] = t12;
 222          3                           c[(i+1)*n+j] = t21;
 223          5                           c[(i+1)*n+(j+1)] = t22;
 224                                  }
 225                                  for (j=j; j<j0+js; j++)
 226                                  {
 227                                      t11 = c[i*n+j];
 228                                      t21 = c[(i+1)*n+j];
 229                                      for (k=k0; k<k0+ks; k++)
 230                                      {
 231                                          t11 = t11 + a[i*n+k]*bt[j*n+k];
 232                                          t21 = t21 + a[(i+1)*n+k]*bt[j*n+k];
 233                                      }
 234                                      c[i*n+j] = t11;
 235                                      c[(i+1)*n+j] = t21;
 236                                  }
 237                              }

Search Engine: (regular expressions supported)
thsub
```

Xprofiler - Application View



```
Program: swim0              Total CPU Usage: 1.31 seconds  (summary of 1 gmon.out profile files)
Display Status: showing  11 out of 95 nodes  and  9 out of 96 arcs
```

Related Work

- **PARAVER (UPC)**
  - Performance Visualization and Analysis Tool
  - Flexibility to represent traces from different environment
  - MPI trace, HW performance counter info (based on PAPI)
- **PAPI (Univ. of Tennessee)**
  - **P**erformance **A**pplication **P**rogramming **I**nterface
  - Design, standardize, and implement a portable and efficient API to access the hardware performance counters

Related Work (continued)

- **TAU (Univ. of Oregon)**
  - **T**uning and **A**nalysis **U**tilities
  - Program and performance analysis tool framework for high-performance parallel and distributed computing
- **mpiP (LLNL/ORNL)**
  - Lightweight profiling library for MPI applications
  - Only collects statistical information about MPI functions
  - With less overhead and have much less data than tracing tools
  - Only uses communication during report generation stage

Challenges

- **Hardware**
  - Different performance counter set
- **Scalability**
  - 64k nodes
  - Tracking communications for each pair:
    - $(65{,}536)^2$x48 bytes = 200 GB
  - Number of processes and events vs. number of screen pixels

Scalability

- **Abstraction**
  - Aggregated performance data
  - Sampling
  - Easy to manage
  - May lose fidelity
- **Performance data organization**
  - Database with one time processing
  - Query, data mining, clustering… etc.
  - Precise information
  - Hard to analyze

Summary

- **IBM High Performance Computing Toolkit**
  - Ported
    - PeekPerf, MP_Profiler
  - Work in progress
    - Xprofiler, HPM
  - Next target
    - Modular I/O: MIO
- **Related Tools**
- **Challenges**
  - Architecture, Scalability

DPOMP

- **Dynamically instruments OpenMP applications**
- **Reports various performance related information**
  - timings, overhead of OpenMP constructs.
- **Modify binaries with performance instrumentation**
  - without requiring access to source codes or recompilation.
- **POMP (Performance monitoring interface for OpenMP) implementation based on dynamic probes**

Simulation Guided Memory Analyzer (SiGMA)

- **Helps to understand precise memory references causing poor memory utilization**
- **Provides fine-grained information useful for**
  - Tuning loop kernels, understanding the cache behavior
- **Consists of**
  - A pre-execution tool that instruments all instructions that refer to memory locations,
  - A runtime data collection tool that performs compression of the stream of memory addresses generated by the instrumentation,
  - Analysis tools that process memory reference trace to provide programmers with tuning information.